VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Podpůrný systém pro správu projektů a úkolů dle metodiky GTD

Support System for Project and Task Management by GTD Metodology

2011 Daniel Dimitrov

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava*.

V Ostravě 15. srpna 2011 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 15. srpna 2011 .....

Rád bych na tomto místě poděkoval Ing. Janu Gaurovi za trpělivost a odborné vedení během mé práce.

Abstrakt

Tato bakalářská práce se zabývá tvorbou informačního systému pro správu projektů a úkolů dle metodiky GTD. V úvodu práce popíši samotnou metodiku, dále se zaměřím na volbu technologií, které budou zapotřebí pro realizaci takového systému, analýzu celého řešení a nakonec implementaci v podobě webové aplikace.

Klíčová slova: Informační systém, webová aplikace, Getting Things Done, GTD, osobní time managment, Python, Django

Abstract

This bachelor thesis deals with the creation of an information system for managing projects and tasks according to GTD methodology. In the introduction I describe the actual methodology, then I focus on the choice of technologies, that will be needed to implement such a system, analysis of the solution and finally the implementation in the form of a web application.

Keywords: Information system, web application, Getting Things Done, GTD, personal time managment, Python, Django

Seznam použitých zkratek a symbolů

GTD	–	Getting Things Done
HTTP	–	Hyper Text Transfer Protocol
URL	–	Uniform Resource Locator
CSS	–	Cascading Style Sheets
HTML	–	HyperText Markup Language
PHP	–	PHP: Hypertext Preprocessor
JS	–	JavaScript
DRY	–	Don't Repeat Yourself
DIE	–	Duplication Is Evil
API	–	Application Programming Interface
SŘBD	–	Systém Řízení Báze Dat

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Žijeme v uspěchaném a moderním věku. Díky vývoji v oblasti technologií jsme schopni cestovat mezi kontinenty rychlostí, o které se našim předchůdcům ani nezdálo, jsme schopni kontaktovat téměř kohokoliv na druhém konci světa během několika málo vteřin, dokážeme odesílat desítky a stovky e-mailů během jediného pracovního dne. Moderní věk nám přinesl mnoho výhod, ale stejně tak nám přinesl nové problémy, které je třeba řešit. Množství informací, které musíme v dnešním světě každodenně zpracovat je obrovské a míra stresu je tomuto množství často úměrná.

Žijeme také v době produktivity práce. Nikdy za celou historii lidstva neexistovala tak obrovská masa lidí, která byla skutečně zodpovědná za svůj pracovní čas. Tato odpovědnost nyní leží na bedrech většiny z nás.

Jak již zadání mé práce napovídá, mým úkolem bylo připravit podpůrný systém pro správu projektů a úkolů dle metody osobního time managmentu Getting Things Done. V první části této práce se budu věnovat metodě jako takové a rozeberu veškeré podrobnosti, které bude potřeba zmínit pro implementaci tohoto systému. Dále se budu věnovat technologiím, které bylo nutné pro jeho vznik využít, analýze celého systému a konečně implementaci, kde poukáži na určité oblasti celého řešení, které byly stěžejním bodem.

2 Osobní time managment

Osobní time managment není novým trendem 21. století, i když by se tak mohlo zdát. Pouze je v naší uspěchané a dynamicky se měnící době více zapotřebí. Hlavní motivací a důvodem, proč se snažit tento trend zařadit do našich životů, vystihují následující dva citáty:

> "Většina stresu nepochází z toho, že toho máme moc. Největší stres vzniká ze všeho započatého a nedokončeného."- Stephen Covey

> "Donuťte člověka pracovat dle přesných předpisů a jeho představivost se rozjede naplno a vytvoří ty nejlepší nápady. Dejte člověku naprostou svobodu a práce se bude vléct donekonečna."- T.S. Eliot

Cílem všech metod osobního time managmentu je maximálně zredukovat stres spojený s vykonávanou prací, zvýšit její produktivitu, pomoct určit její smysl a vytvořit takové podmínky, aby jsme mohli mysl používat kreativně. To vše by se mělo dít za pomocí jednoduchého systému a řady správných návyků.

V současné době je známa celá řada time managmentových metod. Mezi nejznámější patří: 7 návyků skutečně efektivních lidí, jejíž autorem je Stephen R. Covey [1] a Getting Things Done nebo-li GTD, jejíž autorem je americký kouč v oblasti produktivity David Allen [2].

Právě GTD se stalo pro svou jednoduchost a přímočarost velice oblíbenou metodou mezi IT profesionály po celém světě.

2.1 Getting Things Done

Getting Things Done nebo-li GTD objasňuje, jak zvládat dva základní faktory, které jsou neodmyslitelnou součástí každého projektu, či úkolu - kontrolu a perspektivu.

Kontrola je dle Allanova návrhu [3] získávána pěti základními návyky: zaznamenáváním, objasňováním, organizováním, reflexí a samotnou akcí, která vede ke splnění daného úkolu. Jedná se o faktor, který lze získat systematizací pracovních postupů a procesů a je to tedy část celého systému, která je převážně předmětem této práce.

Perspektiva je pak získávána a udržována pomoci stanovení: projektů, oblastí zodpovědnosti, cílů, vizí, smyslů a hodnot.

2.1.1 Zaznamenávání

Zaznamenávání (občas také označováno jako shromažďování) je prvním návykem pro získání kontroly. GTD zavádí tzv. vstupní schránky [2], do kterých je nutné během dne zaznamenávat všechny naše závazky. Obecně je doporučeno mít vstupní schránky dvě. Jednu pro fyzické předměty jako jsou např. různé materiály v papírové podobě, či jakékoliv jiné předměty a druhou v elektronické, či v papírové podobě, do které si během dne zaznamenáváme veškeré úkoly, povinnosti, nápady a další závazky, které nám přijdou na mysl.

Při nasazování celé metodiky je tato fáze obvykle nejdelší. V první části je nutné projít veškeré nezpracované materiály, shromáždit veškeré předměty, které poutají naši pozornost a jsou spjaty s nějakou akcí, projít veškerou e-mailovou korespondenci, shromáždit nevyřízenou fyzickou poštu apod.

Druhá část spočívá v úklidu naší mysli. Je nutné ji provádět ideálně v naprostém klidu a bez vyrušování. Celý proces spočívá v tom, položit si před sebe na prázdný stůl pouze papír a tužku a v mysli procházet veškeré oblasti našeho života, kde je možné, že musíme vykonat nebo jsme zodpovědní za určitou činnost. Jedná se o oblasti jako je práce, konkrétní projekty, jednotliví klienti, také např. vize, kterých by jsme chtěli dosáhnout, naše rodina apod. Výsledkem této části, která může trvat i několik hodin by měl být seznam všech našich závazků, který jsme do této chvíle uchovávali pouze v paměti.

V průběhu osvojování si celé metodiky je v této fázi klíčové vytvořit si návyk, díky kterému budeme do vstupní schránky skutečně neustále umisťovat veškeré závazky, nápady a podněty, které během dne vyvstanou nebo nám zrovna přijdou mysl.

Tento návyk je první z řady návyků, které Allen označuje jako klíčové k tomu, aby jsme dosáhli stavu: mysl jako voda [2].

Stejně klíčové je se s těmito závazky poté dále vypořádat ve fázi objasňování a organizace. Toto vypořádávání je dle Allena nutné provádět alespoň jedenkrát v průběhu jednoho až dvou dnů [2].

Dle mých dlouhodobých zkušeností je neplnění této fáze první ze dvou nejzákladnějších chyb, díky které se následně celý systém bortí a stává se nepoužitelným.

2.1.2 Objasňování a organizování

Druhou fází pro získání kontroly je objasňování a organizace. Na začátku této fáze obsahuje fyzická, elektronická či papírová vstupní schránka několik položek a nyní je nutné rozhodnout, jak s nimi naložit.

Během této fáze zavádí GTD několik nových pojmů:

- Další krok - další krok je jakákoliv fyzická činnost nebo úkol, který je již dále nedělitelný a jedná se o první potřebný krok, který je nutné splnit pro to, aby se dala celá záležitost do pohybu. Tento krok musí být jasně definován.

  Příklad 2.1
  Mějme například záležitost - jít do kina na premiéru nového filmu. Další logický krok v této záležitosti je - zvednout telefon a lístky zarezervovat. Další krok by samozřejmě mohl být rozdělen ještě dále na jednotlivé kroky jakou jsou: vzít telefon, najít číslo, zatelefonovat, zarezervovat lístky. V tomto ohledu by se ovšem jednalo o micro management, před kterým GTD důrazně varuje a který vede pouze k nepřehlednosti celého systému a je značně neefektivní také z pohledu vynaloženého času na údržbu. Zvednout telefon je tedy krok, který je již dále nedělitelný, jedná se o první potřebný krok, který je nutné provést a zároveň je dostatečně jasně definován, aby bylo zřejmé co obnáší. Tento krok také vyžaduje jasnou fyzickou akci,

což je jedna ze základních vlastností, které je nutné dodržet. Krok typu zamyslet, se do jakého kina jít, by byl špatným dalším krokem, který by vedl pouze k odkládání celé záležitosti a prokrastinaci. ■

- Projekt - je seskupení dalších kroků, které vedou k jednomu cíli, který je jasně definován a po jehož dokončení je projekt označen jako vyřízený.

  Příklad 2.2
  Záležitost z výše uvedeného příkladu 2.1 je vzorovým projektem skládajícího se například z kroků: zarezervovat lístky, oznámit termín přítelkyni, jít do kina. ■

- Seznam projektů - seznam projektů je elektronický, či papírový seznam všech našich projektů, které jsme schopni momentálně řešit.

- Seznam čekám na - na seznam čekám na jsou zaznamenány veškeré kroky, které byly delegovány na jinou osobu a za jejichž vyřízení jsme zodpovědní.

- Seznam dalších kroků - na seznamu dalších kroků jsou vedeny jednotlivé kroky, které je nutno provést proto, aby se dané záležitosti daly do pohybu.

- Kalendář - kalendář je místo, na které jsou zaznamenány veškeré kroky a projekty, které jsou časově ohraničeny nebo které musí být provedeny v určitou dobu.

  Příklad 2.3
  Z našeho příkladu 2.2 se jedná o krok - jít do kina, který je vázán na určité datum a čas. ■

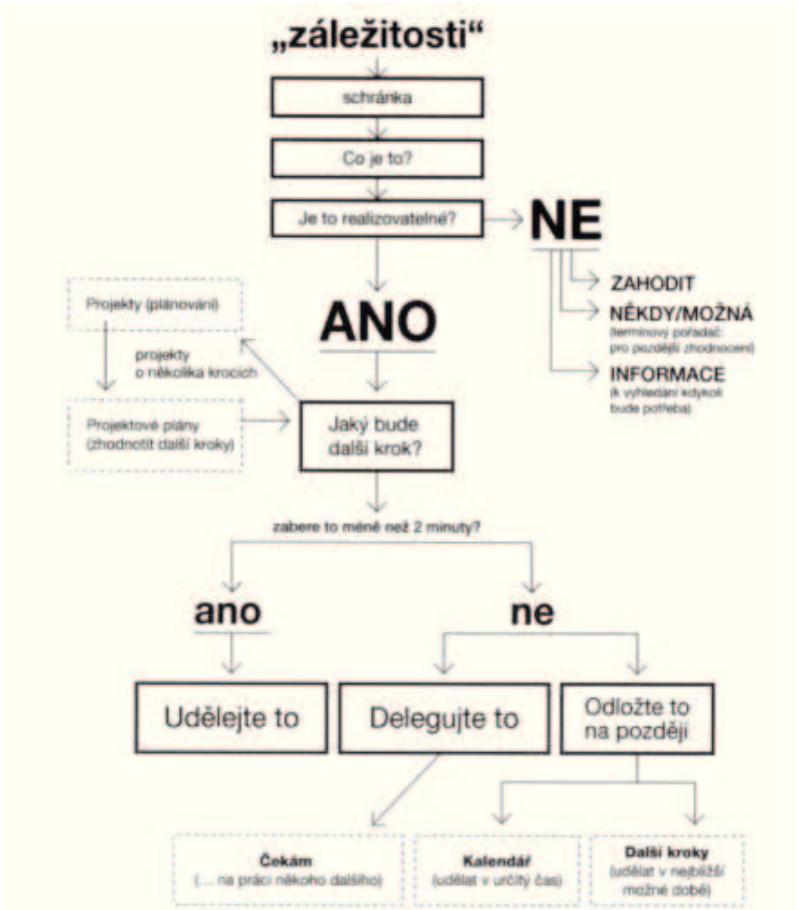
- List někdy/možná - na tomto listě jsou vedeny veškeré další kroky, projekty, nápady a náměty, které nejsou momentálně realizovatelné nebo na které nezbyl čas pro realizaci, ale je možné, že budou v budoucnu aktuální.

- Kontext - kontext je osoba, místo nebo zařízení, které je spjato s daným krokem a bez kterého není možné tento krok splnit. V modernějších GTD systémech lze také jednotlivé kroky dle kontextu filtrovat, což vede k větší produktivitě práce.

  Příklad 2.4
  V našem příkladě 2.2 by měl být krokům: zarezervovat lístky a oznámit termín přítelkyni přidělen kontext telefon. Druhý z uvedených kroků by poté mohl mít přidělen mimo jiné kontext přítelkyně. ■

Celý proces objasňování a organizování v této fázi se řídí dle GTD diagramu [2], který demonstruje obrázek 1.

Na začátku je třeba určit, zda je daná záležitost vůbec realizovatelná. Na základě tohoto rozhodnutí následuje jeden z následujících modelů:



Obrázek 1: GTD Diagram

- V případě, že je realizovatelná, je zapotřebí určit další krok. Záležitost se v této fázi samozřejmě může stát sama o sobě dalším krokem, či projektem, či být zařazena již do existujícího projektu. V každém z těchto příkladů je nutné záležitost zavést na příslušný list. V případě, že se záležitost stala projektem, je nutné identifikovat všechny blízké kroky vedoucí k jeho dokončení. Jakmile se tak stane, je zapotřebí zaměřit se na každý krok v projektu (či pouze daný další krok, pokud je záležitost krokem sama o sobě) a položit si otázku: Zabere mi splnění tohoto úkolu méně než dvě minuty? Pokud ano, provedeme úkol ihned. Pokud ne a v případě, že existuje oprávněnější osoba, která by měla úkol řešit, tak jej delegujeme a poznamenáme si jej na seznam úkolů - čekám na. Pokud úkol nemůžeme delegovat, tak jej zaznamenáme na seznam dalších kroků k danému projektu, popřípadě, pokud je spojen s konkrétním datem, poznamenáme do kalendáře a přidělíme mu kontext.
- V případě, že není záležitost realizovatelná, následují tři možnosti řešení:
  - Záležitost si uložíme jako informaci pro pozdější využití.

- Záležitost si zapíšeme na list někdy/možná, protože momentálně nejsme schopni s její realizací nic udělat, ale v budoucnu se může tato situace změnit.
- V případě, že nevyužijeme ani jednu z výše zmíněných možností, nezbývá nic jiného než záležitost zahodit.

Výše popsaný koncept je pro GTD v této fázi zásadní a celý systém je založen na jeho architektuře. Osvojování probíhá překvapivě velice rychle a to díky skutečnosti, že je celý koncept založen na modelu přirozeného plánování [2]. Allen jednoduše celý tento koncept připravil na modelu, který lidský mozek využívá již od nepaměti. Zavedl pouze nové pravidlo dvou minut tak, aby eliminoval jakékoliv další náznaky micro managmentu, nebo-li dělání time managmentu pro time managment a nikoliv pro zefektivnění skutečné práce a zabránil tak také přehlcenosti a nepřehlednosti celého systému. Dále nás Allen nově nutí veškeré úkoly zapisovat na externí seznamy, díky kterým získáváme lepší přehled o svých závazcích, snižujeme míru vnitřního stresu, přestáváme hlavu používat jako datovou úschovnu a začínáme ji používat ke kreativnímu myšlení.

2.1.3 Reflexe

V této chvíli se systém skládá z několika vytvořených listů, na kterých jsou zaznamenány různorodé úkoly k řešení, jsou zřízeny vstupní schránky, do kterých umísťujeme a zaznamenáváme nově vzniklé závazky a byl objasněn upravený model přirozeného plánování.

Nyní je nutné osvojit si efektivní návyk pro to, aby byl celý systém neustále aktuální. V opačném případě by se postupem času zhroutil podobně jako domeček z karet.

V této fázi Allen zavadí tzv. týdenní hodnocení [2]. Týdenní hodnocení je jednoduchý proces, který je nutné vykonávat s týdenní periodicitou a skládá se z následujících kroků:

- Revize vstupních schránek - je zapotřebí projít veškeré vstupní schránky a zorganizovat jejich obsah pomocí diagramu z kapitoly 2.1.2. Dále je nutné promyslet, zda se v průběhu týdne nevyskytly nějaké závazky, které nebyly do schránky poznamenány. Jako pomůcku v této fázi brainstormingu Allen doporučuje projít materiály, které jsme v průběhu týdne nastřádali na pracovním stole.

- Revize listu projektů - dále je vhodné projít celý seznam se zaznamenanými projekty. Je nutné projít všechny kroky k těmto projektům a zhodnotit, zda projekt neobsahuje kroky, které jsou již splněny nebo již nejsou aktuální, dále je třeba rozhodnout, zda se v průběhu týdne nevyskytly skutečnosti, díky kterým jsme nuceni projekt odložit nebo se ho úplně zbavit.

- Revize ostatních míst systému - pokud jsou zrevidovány veškeré vstupní schránky a list s projekty, je zapotřebí zrevidovat veškeré další listy a úložná místa systému. V této fázi tedy procházíme listy čekám na, někdy/možná, list dalších kroků, kalendář a archiv, do kterého se ukládají informace. Opět se vybírá položka po položce a rozhoduje se, zda je stále aktuální, zda je potřebná a na základě tohoto rozhodnutí se s ní naloží.

Reflexe je předposledním z návyků, které GTD učí a který je pro dlouhodobý chod systému, jeho použitelnost a aktuálnost klíčový.

Z dlouhodobých zkušeností vyplývá, že nedodržování tohoto návyku je druhou nejčastější chybou, která vede k pádu celého systému.

2.1.4 Akce

Úkolem poslední fáze je naučit se ten nejdůležitější návyk, bez kterého by bylo vše předchozí k ničemu. Posledním návykem k osvojení je vykonávat zvolené kroky správně a ve správném pořadí. Celý systém je postaven na myšlence usnadnit a zefektivnit práci a akce je jeho poslední klíčovou částí.

Dle Allena [2] je třeba v této části přejít z intuitivního rozhodování o volbě úkolu k jasně vytyčenému systému, který za nás dokáže určit, jaký úkol je nutné momentálně řešit.

Při výběru vhodného kroku k řešení je nutné volit na základě těchto čtyř kritérií, které jsou seřazeny dle priority:

- Kontext - kroky s jakým kontextem je v mé moci momentálně řešit? Mám u sebe mobilní telefon? Notebook? Jsem připojen k internetu? Tyto otázky je nutné zodpovědět. Práce s kontexty je jednou z největších výhod GTD a GTD je jediný time managmentový systém, který kontexty jako takové zavádí. V praxi efektivní práce s kontexty probíhá tak, že je vybrán jeden kontext - např. telefon - a veškeré kroky, které tento kontext obsahují, jsou plněny jeden za druhým. Vzhledem k tomu, že jde o opakující se činnost, dochází většinou k velké úspoře času v porovnání s řešením několika různorodých kroků.

- Dostupný čas - v případě, že je kontext vyhovující, je dostupný čas druhým nejdůležitějším kritériem. Pokud pracujeme v kontextu telefon a jsme například v situaci, kdy nám začíná do 5 minut porada, tak zřejmě nesplníme úkol, na základě kterého je nutné informovat důležitého klienta o nových krocích v projektu, který se ho týká, ale dáme přednost nějakému jednoduššímu telefonátu, který můžeme rychle odbýt.

- Dostupná energie - pokud jsme zvolili kontext a máme na úkol dostatek časového prostoru, je dostupná energie předposlední kritérium při výběru vhodného úkolu. V této fázi je nutné zhodnotit, zda na daný úkol máme dostatek energie. V pozdních odpoledních hodinách po náročném dni se tak raději nepustíme do stěžejního úkolu a vyřídíme raději několik menších.

- Priorita daného úkolu - priorita je poslední kritériem při volbě úkolu. V této fázi zhodnocujeme, zda neexistuje v našem systému úkol ve stejném kontextu, který zabere relativně stejné množství času a energie a který má větší prioritu. Pokud ano, překročíme k tomuto úkolu, v opačném případě realizujeme, již zvolený.

Celý výše uvedený koncept opět vychází z přirozeného plánování a podvědomě jej v určité variaci provádí každý z nás. Vytvoření návyku, aby byl vykonáván ve správném pořadí, je posledním předpokladem pro efektivní funkci celého systému.

Akce je poslední fází GTD a objasněním této poslední části jsem uzavřel pojednání o celém systému. Mým úkolem bylo vytvořit informační systém dle určitých specifikací, který bude z těchto zásad vycházet. Jedním z prvních kroků po objasnění problému je volba jednotlivých technologií, na kterých bude celé řešení postaveno a tomuto problému se proto také věnuji v následující kapitole.

3 Metodiky a technologie

3.1 Volba jazyka

Na začátku každého projektu je vývojář postaven před důležitou otázku: Jaký programovací jazyk je vhodné zvolit pro řešení tohoto problému? Je zřejmé, že různé jazyky přinášejí různé výhody, které se dají využít u specifických typů projektů. Co se webových aplikací týče, vynecháme-li možnosti jako je čistě statický web a robusní Web 2.0 aplikace s notnou dávkou JavaScriptu a zaměříme se pouze na serverové technologie, má v dnešní době vývojář ve své podstatě pouze několik základních možností.

V této otázce sehráli roli osobní preference. Vždy jsem byl příznivcem skriptovacích jazyků a nejsem příznivcem momentálního mainstreamu, kterým je jazyk PHP.

Na základě dohody s garantem mé bakalářské práce panem Ing. Janem Gaurou jsem se nakonec rozhodl aplikaci implementovat v jazyce Python s použitím frameworku Django [4].

3.2 Django

Obecně se dá framework charakterizovat jako rozsáhlá knihovna, která programátorovi pomáhá řešit běžné problémy v dané oblasti pomocí propůjčení vhodných postupů a již implementovaných metod.

Cílem každého frameworku a důvodem, proč framework použít, je bezesporu zvýšení efektivity vývojářovi práce, což je nejen výhodné pro vývojáře, ale také pro zadavatele. Další nespornou výhodou je ucelená koncepce. Frameworky programátorovi často vytváří mantinely, mezi kterými je nucen se držet. Tato vlastnost nejenže opět zvyšuje efektivitu, ale usnadňuje také orientaci dalších případných vývojářů, kteří jsou nuceni projekt spravovat nebo rozvíjet později. V tomto ohledu mohou být frameworky obrovským přínosem zejména pro mladé vývojáře, které tak od začátku vedou správným směrem a učí je používat vhodný objektový model.

Django je open source framework naprogramovaný v jazyce Python. Tak jako většina moderních webových frameworků je postaven na MVC architektuře a DRY principech.

3.3 MVC

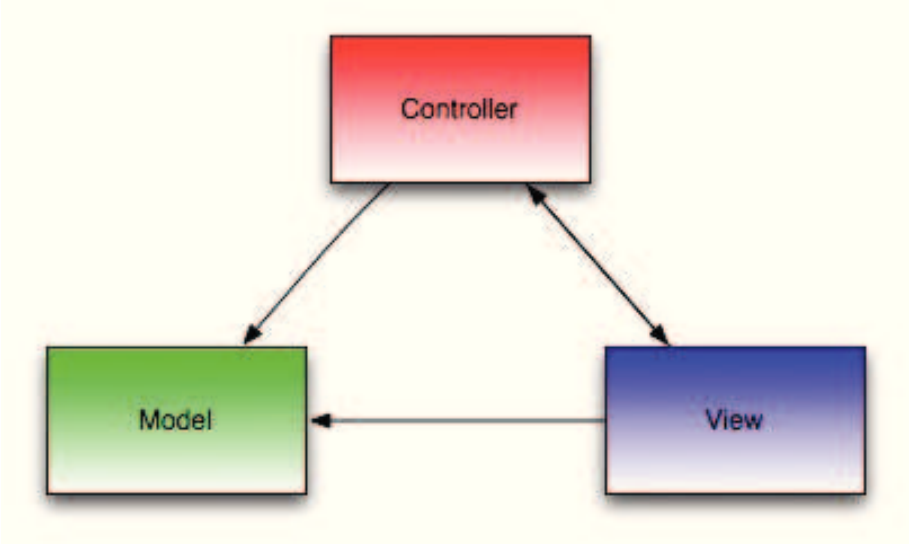
V roce 1979 byla světu představena nová architektura pro vývoj interaktivních aplikací s názvem MVC. Trygve Reenskaug, autor této architektury, v tomto roce zveřejnil model dělící aplikaci do tří základních částí - Model, View a Controller.

MVC je dnes nedílnou součástí většiny moderních webových aplikací. Mnohdy je vnímáno jako jeden z mnoha návrhových vzorů, což není zcela přesné. MVC je skutečně spíše nutné vnímat jako architektonickou část softwaru než jako klasický návrhový vzor.

Důvodem vzniku MVC byla potřeba oddělit jednotlivé logické části aplikace tak, aby je bylo možné upravovat zcela samostatně a to bez žádného, či minimálního vlivu na další logické části. Jak je již uvedeno výše, tyto logické části se nazývají Model, View a Controller. Funkcí Modelu je reprezentovat data a zajišťovat business logiku, View pak

zobrazuje uživatelské rozhraní aplikace a úkolem Controlleru je zajistit aplikační logiku a tok událostí v aplikaci [5].

MVC se využívá v mnoho variantách a variacích. Dnes běžně využívaný model mnoha webovými frameworky je MVC architektura představená ve Smalltalku [6], kterou demonstruje obrázek 2.



Obrázek 2: MVC

3.3.1 Model

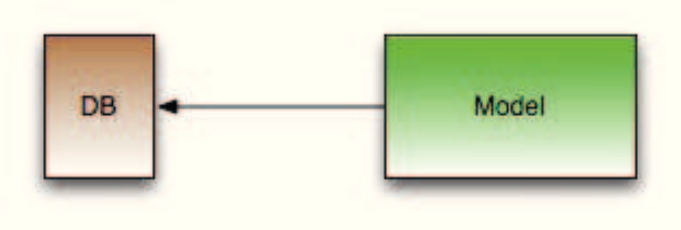
Model je společně s Controllerem část aplikace, ve které se z pohledu architektury nejčastěji chybuje. Důvodem je fakt, že princip MVC jako takový je velice jednoduchý. Skutečně pochopit celou architekturu do detailu a dokázat určit, zda tato část logiky patří spíše do Controlleru nebo do Modelu, zda vazby, které jsou v aplikaci vytvořeny jsou skutečně správně, je otázkou mnoha zkušeností a mnohdy i let. Stěžejní částí je samozřejmě fakt, že celá aplikace není vybudována pouze na jednom Modelu, Controlleru a View. V praxi se aplikace skládá z velkého množství těchto logických prvků, které na sebe různě navazují.

Úkolem Modelu je udržovat data o aplikaci a starat se o veškerou logiku, která s daty souvisí. Příkladem této logiky může být např. validace dat před uložení do datového uložiště, celková komunikace s uložištěm, zpracovávání výstupu dat, které jsou dále předány Controlleru pro zobrazení do View apod.

Model se z pohledu vazeb vždy a to v jakékoliv modifikaci MVC, ať již v desktopové, či webové aplikaci chová zcela samostatně. Nevytváří v žádném případě vazby na zbývající dvě logické části aplikace.

Příklad 3.1
Mějme například za úkol vytvořit část webové aplikace pro správu uživatelů. Pro uživatele je v našem datovém uložišti vytvořena tabulka User. Úkolem Modelu poté bude zajis-

tit veškerou komunikaci s datovým uložištěm, bude zodpovědný za validaci údajů při vložení nového, či editaci stávajícího uživatele a za poskytnutí odpovídajících dat Controlleru při požadavku na zaslání těchto dat. Model bude tvořit samostatná třída, která nebude mít vazbu na žádný Controller, či View. Bude mít ovšem vazbu na tabulku z uložiště, se kterou bude nucen komunikovat. Tento Model je demonstrován obrázek 3. ■



Obrázek 3: Model

3.3.2 Controller

Controller zajišťuje veškerou aplikační logiku a tok událostí v aplikaci. Jeho standardní prací je obdržet a popřípadě zpracovat vstup pocházející z View, předat informace příslušnému Modelu, na který má vazbu, zaslat požadavek Modelu na předání určitých dat, informovat View o tom, že má být překresleno, předat View před vykreslení požadovaná data apod.

V případě webové aplikace jsou veškeré vstupy zasílány prostřednictvím HTTP požadavků a URL adres.

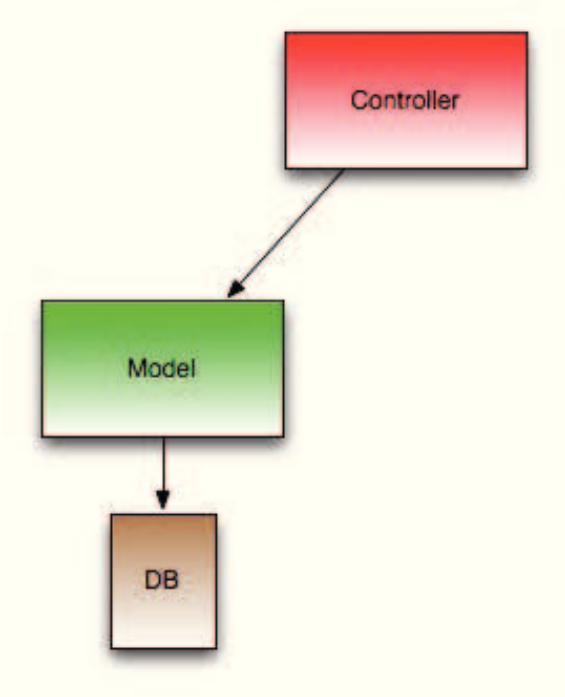
Příklad 3.2
Z našeho předchozího příkladu 3.1 by bylo nutno v Controlleru aplikovat logiku pro přidání, aktualizaci a smazání konkrétního uživatele. Dále by bylo nutné aplikovat logiku pro komunikaci s jednotlivými View a vytvořit vazbu na konkrétní Model. Takto rozšířenou architekturu demonstruje obrázek 4. ■

3.3.3 View

View je představitelem uživatelského rozhraní celé aplikace. Jeho úkolem je vhodně interpretovat data získaná prostřednictvím Controlleru a zároveň je jedinou vstupní branou, díky které aplikace získává data z vnějšího světa.

Ve webovém prostředí je View nejčastěji tvořeno HTML kódem a různými CSS styly, které jsou zaslány prohlížeči jako odpověď na HTTP požadavek určité URL adresy.

Příklad 3.3
Pokud opět navážeme na výše uvedený příklad 3.2, bude nutné implementovat hned několik View. View pro zobrazení konkrétního uživatele, View pro editaci stávajícího



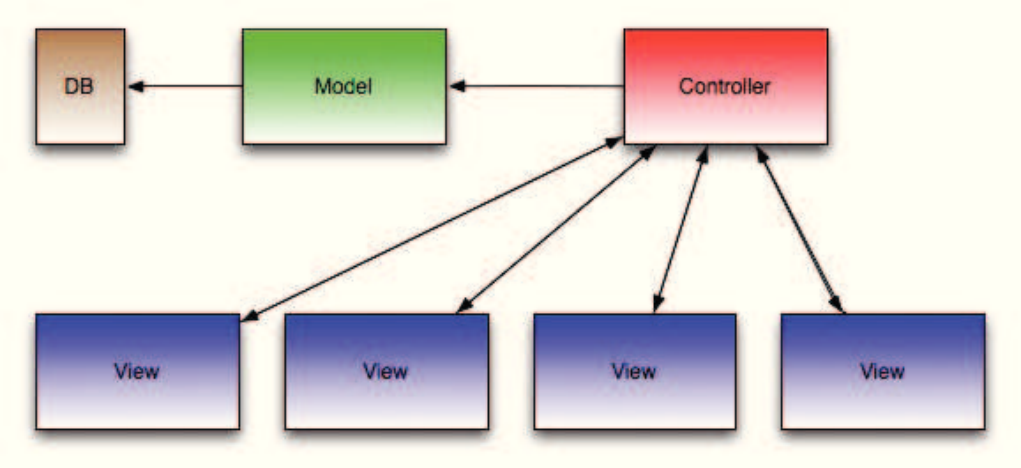
Obrázek 4: Controller a Model

a přidání nového uživatele a konečně View pro zobrazení všech existujících uživatelů. Všechny tyto View poté budou mít vazbu na Controller, kterému zasílají určité požadavky a který je zodpovědný za jejich vykreslování. Konečnou architekturu demonstruje obrázek 5. ■

3.3.4 Rozdíly v MVC architektuře a další vyjímky

Dle obrázků 2 a 5 je patrné, že při standardním vývoji webové aplikace může docházet k menším rozdílům ve filozofii a odklonění od klasického MVC modelu. MVC architektura demonstrována obrázkem 5 je klasickou architekturou v případě, že by aplikace byla tvořena pomocí frameworku Ruby on Rails [8] a programovacího jazyka Ruby. Je samozřejmě zcela běžné, že jednotlivé frameworky a prostředí využívají různých modifikací původní MVC architektury. Základní principy jsou ovšem většinou zachovány. V případě webových aplikací je jediným větším rozdílem oproti původnímu MVC nevytváření přímé vazby mezi Modelem a View. Veškerá komunikace mezi těmito logickými celky probíhá standardně prostřednictvím Controlleru. V tomto ohledu podobný princip využívá např. Cocoa Framework [9] pro tvorbu iOS a OS X aplikací [10].

Zajímavé modifikace do tohoto modelu vnášejí robusní Web 2.0 aplikace tvořeny z velké částí Javascriptovým kódem na straně klienta. Díky JS se velká část aplikační logiky přenáší ze strany serveru na stranu klienta. Příkladem takové aplikace je například e-mailová služba GMail [7] společnosti Google. V tomto ohledu je pak potřeba implemen-



Obrázek 5: Celková architektura

tovat MVC architekturu také na straně klienta, což ovšem není případem této bakalářské práce.

3.3.5 DRY

V souvislosti s MVC by bylo také vhodné zmínit princip, ke kterému celá architektura vybízí. Jedná se o DRY nebo-li Dont Repeat Yourself, který je občas označován také jako DIE z anglického Duplication is Evil. Jak už název napovídá, jedná se o princip využíván v oblasti softwarového vývoje, jehož hlavním cílem je pokud možno eliminovat veškeré opakující se informace.

Výhody při aplikování tohoto principu jsou zcela zřejmé: nedochází k redundanci, zvyšuje se přehlednost samotného kódu, při nutnosti jakékoliv změny stačí tuto změnu provést pouze na jednom místě tak, aby ovlivnila všechny objekty, které s ní souvisí.

3.4 Volba databázové technologie

Stejně jako při volbě programovacího jazyka, je i volba správného SŘBD velice důležitou součástí každého projektu. Díky architektuře Django frameworku není přechod mezi jednotlivými ŠRBD naštěstí nijak problematický.

Ze specifikace projektu je zřejmé, že celkový objem dat nebude nikterak velký, a proto jsem se nakonec rozhodl použít malou a rychlou knihovnu SQLite [11].

4 Analýza a návrh informačního systému

Před začátkem práce na celém systému bylo nejprve zapotřebí celý problém analyzovat. Úkolem analýzy je získat detailnější představu o všech problémech, které bude nutno řešit, zjistit přibližný rozsah prací a připravit efektivní kostru aplikace, které se mohou vývojáři následně držet.

Analýza problému ve své podstatě začala již při studiu GTD problematiky. Než se z této fáze přesuneme k návrhu samotné struktury systému, je třeba zvážit a pečlivě analyzovat veškeré požadavky, které jsou na systém kladeny. V následujících podkapitolách jsou specifikovány veškeré funkční a nefunkční požadavky na systém a nakonec graficky znázorněny obrázkem 6 pomocí use case diagramu.

4.1 Funkční požadavky

Funkční požadavky vycházejí ze zadání bakalářské práce a také ze samotné analýzy GTD systému. Veškeré požadavky jsou specifikovány pro roli uživatele.

Po spuštění systému by měl mít uživatel tyto možnosti:

- Vytvořit a editovat úkol, vyplnit jeho údaje jako je název, termín dokončení, zda se jedná o úkol k dnešnímu dni a poznámku k úkolu.
- Přiřadit k úkolu neomezený počet štítků pro kontext.
- Pohodlně přesouvat úkol mezi jednotlivými projekty a listy.
- Označit úkol jako splněný.
- Smazat daný úkol.
- Vytvořit a editovat projekt, vyplnit jeho údaje jako je název, termín dokončení, zda se jedná o projekt k dnešnímu dni a poznámku k projektu.
- Seskupovat jednotlivé úkoly do projektů.
- Označit projekt jako splněný.
- Smazat projekt.
- Zobrazit v přehledné formě všechny listy systému.
- Filtrovat v jednotlivých listech, či projektech úkoly dle kontextu.
- Nastavit přístup ke svému Google účtu pro import údajů z kalendáře.
- Zobrazovat kalendář.
- Smazat událost vytvořenou v kalendáři.

4.2 Nefunkční požadavky

Informační systém by měl splňovat trendy moderní webové aplikace. Systém bude nasazen na privátním serveru s přístupem na internet.

4.3 Use case diagram

Pro vizualizaci výše popsaných požadavků je vhodné použít use case diagram, který je demonstrován obrázkem 6.

4.4 Seznam listů, projektů a logistika úkolů v rámci systému

Závěrem je třeba specifikovat seznam listů a logistiku, kterou jsou dané úkoly v systému do listů a projektů řazeny. Celý logistický proces vychází ze zásad systému GTD.

4.4.1 Seznam listů

Listy představují jednotlivé seznamy dle metodiky GTD, které byly detailně objasněny v kapitole 2.1.2.

- Vstupní schránka (Inbox).
- List aktuálních událostí a úkolů, které je nutno vykonat v daný den (Today).
- List dalších kroků k daným projektům (Next).
- Kalendář (Scheduled).
- List někdy/možná (Someday).
- List čekám na je realizován pomocí kontextů, kdy má uživatel možnost vytvořit kontext s názvem čekám a přiřadit si jej k jednotlivým úkolům.
- List aktuálních projektů (Projects).
- List dokončených úkolů (Logbook).

4.4.2 Logistika úkolů

Níže je specifikována interní logistika v systému, který vychází ze zásad GTD.

- Po vytvoření nového úkolu je úkol zařazen do Inboxu.
- V Inboxu jsou veškeré úkoly, které nejsou přiřazeny k žádnému projektu nebo nejsou momentálně aktuální - není u nich nastaven příznak, že se jedná o úkol nutný realizovat během aktuálního dne a nemají nastaveno datum dokončení na daný den.

- Úkol je možné z Inboxu přiřadit do jakéhokoliv z následujících listů: Someday, Today, Projects - zde je nutno úkol přiřadit ke konkrétnímu projektu.
- V listu Today jsou zobrazeny veškeré úkoly, u kterých byl nastaven příznak today, dále úkoly, jejichž datum dokončení je totožné s aktuálním datem a nakonec události z kalendáře, které jsou chronologicky řazeny.
- V listu Next jsou zobrazeny aktuální projekty a k nim všechny přiřazené úkoly.
- V listu Someday jsou zobrazeny veškeré úkoly, které byly do tohoto listu ručně přiřazeny.
- V Scheduled jsou zobrazeny veškeré události z kalendáře a jsou chronologicky řazeny.
- V listu Projects jsou zobrazeny veškeré aktuální projekty.
- U každého úkolu je možné nastavit příznak dokončeno a úkol je automaticky přesunut do listu Logbook.
- U každého projektu je možné nastavit příznak dokončeno a veškerým úkolům z daného projektu je přiřazen příznak dokončeno, jsou přesunuty do listu Logbook a projekt není nadále zobrazován v listu Projects.
- V listu Logbook jsou zobrazeny veškeré úkoly, které byly označeny jako splněné.
- U každého úkolu v listu Logbook je možné nastavit příznak nedokončeno a úkol je přesunut zpět na list, na kterém se nacházel. V případě, že se úkol nacházel v některém z projektu, je přesunut zpět do projektu, který je opět zobrazován v listu Projects.

4.5 Datová analýza

V této fázi známe veškeré funkční a nefunkční požadavky na systém, máme představu o možnostech, které mají být uživateli nabídnuty a máme objasněnu vnitřní logistiku celého systému.

Díky výše uvedeným informacím tak můžeme přikročit k analýze datové části celého systému, kde budou objasněny vztahy mezi jednotlivými objekty v databázi a struktura jednotlivých tabulek.

4.5.1 Konceptuální model

Díky konceptuálnímu modelu je možné popsat objekty a vztahy mezi objekty v databázi. Na obrázku 7 je za pomocí jazyka UML konceptuální model systému vizualizován, dále poté následuje slovní popis jednotlivých tříd a jejich atributů.

Třída Project reprezentuje projekt v systému a je třeba u něj evidovat následující atributy:

- unikátní identifikátor
- název projektu
- čas a datum vytvoření projektu a jeho poslední aktualizace
- poznámky k projektu
- termín dokončení projektu
- nastavení pro zobrazení všech úkolů projektu v Today listu
- stav projektu

Třída Task reprezentuje úkol v systému a je třeba u ní evidovat následující atributy:

- unikátní identifikátor
- název úkolu
- čas a datum vytvoření úkolu a jeho poslední aktualizace
- poznámky k úkolu
- termín dokončení úkolu
- nastavení pro zobrazení úkolu v Today listu
- nastavení pro zobrazení úkolu v Someday listu
- stav úkolu

Třída Tag reprezentuje kontexty v systému, které je možné přiřazovat jednotlivým úkolům a je třeba u ní evidovat následující atributy:

- unikátní identifikátor
- název kontextu
- pořadí kontextu, ve kterém má být zobrazován

Třída Settings reprezentuje v systému nastavení pro připojení ke Google účtu a je třeba u ní evidovat následující atributy:

- unikátní identifikátor
- e-mailovou adresu, která je spojena s Google účtem
- heslo k účtu

Název	Datový typ	Klíč	NULL	Default
id	int	P	NE	-
name	varchar(200)	-	NE	-
createdat	datetime	-	NE	-
updatedat	datetime	-	NE	-
notes	text	-	ANO	-
duedate	date	-	ANO	-
today	boolean	-	NE	FALSE
someday	boolean	-	NE	FALSE
checked	boolean	-	NE	FALSE

Tabulka 1: Project

Název	Datový typ	Klíč	NULL	Default
id	int	P	NE	-
projectid	int	C	ANO	-
name	varchar(200)	-	NE	-
createdat	datetime	-	NE	-
updatedat	datetime	-	NE	-
notes	text	-	ANO	-
duedate	date	-	ANO	-
today	boolean	-	NE	FALSE
someday	boolean	-	NE	FALSE
checked	boolean	-	NE	FALSE

Tabulka 2: Task

Název	Datový typ	Klíč	NULL	Default
id	int	P	NE	-
name	varchar(200)	-	NE	-
ord	int	-	NE	-

Tabulka 3: Tag

Název	Datový typ	Klíč	NULL	Default
id	int	P	NE	-
tagid	int	C	NE	-
taskid	int	C	NE	-

Tabulka 4: TagToTask

Název	Datový typ	Klíč	NULL	Default
id	int	P	NE	-
mail	varchar(100)	-	NE	-
password	varchar(100)	-	NE	-

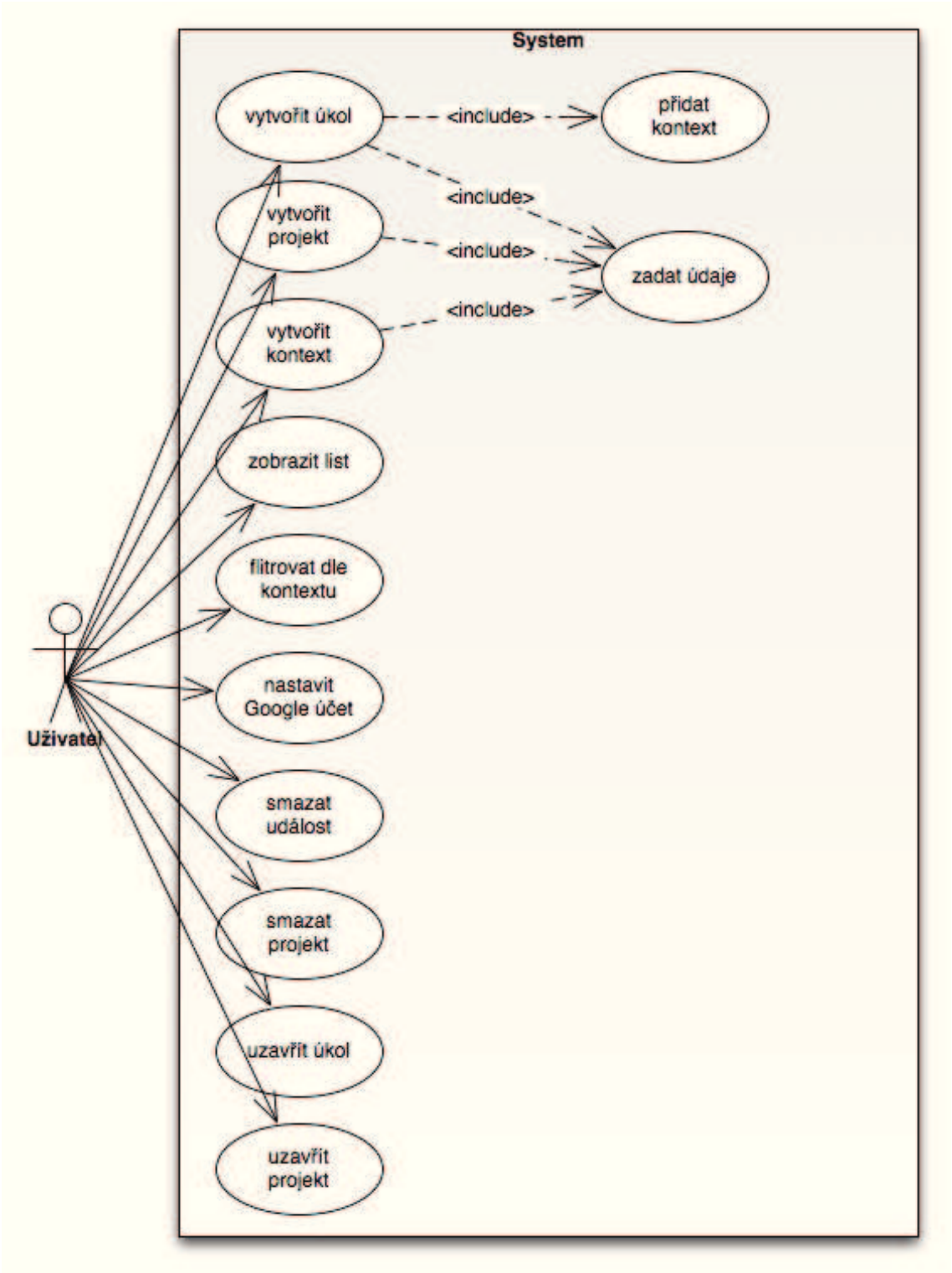
Tabulka 5: Settings

4.5.2 Datový slovník

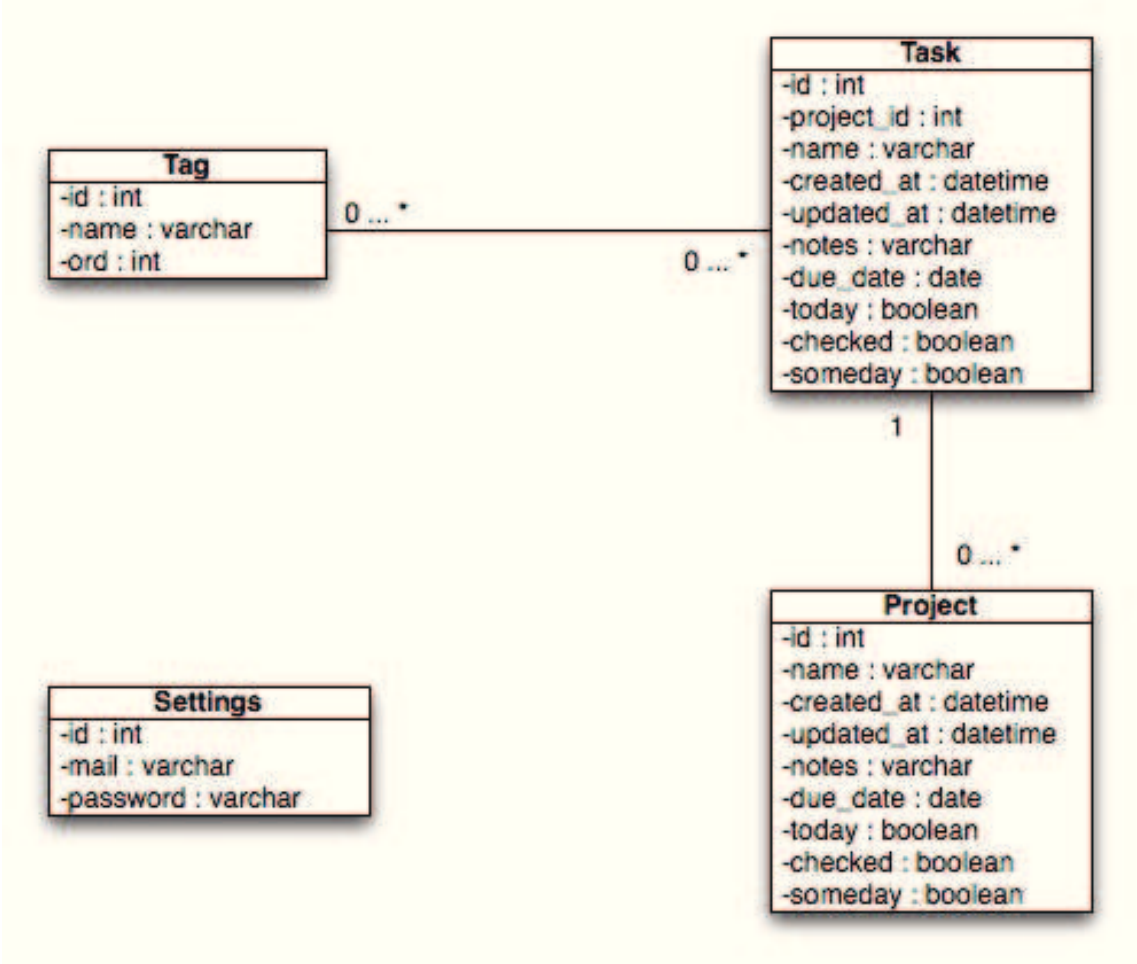
Atributy jednotlivých entit jsou detailně specifikovány v datovém slovníku, který reprezentují tabulky 1, 2, 3, 4, a 5.

4.6 Výstup analýzy

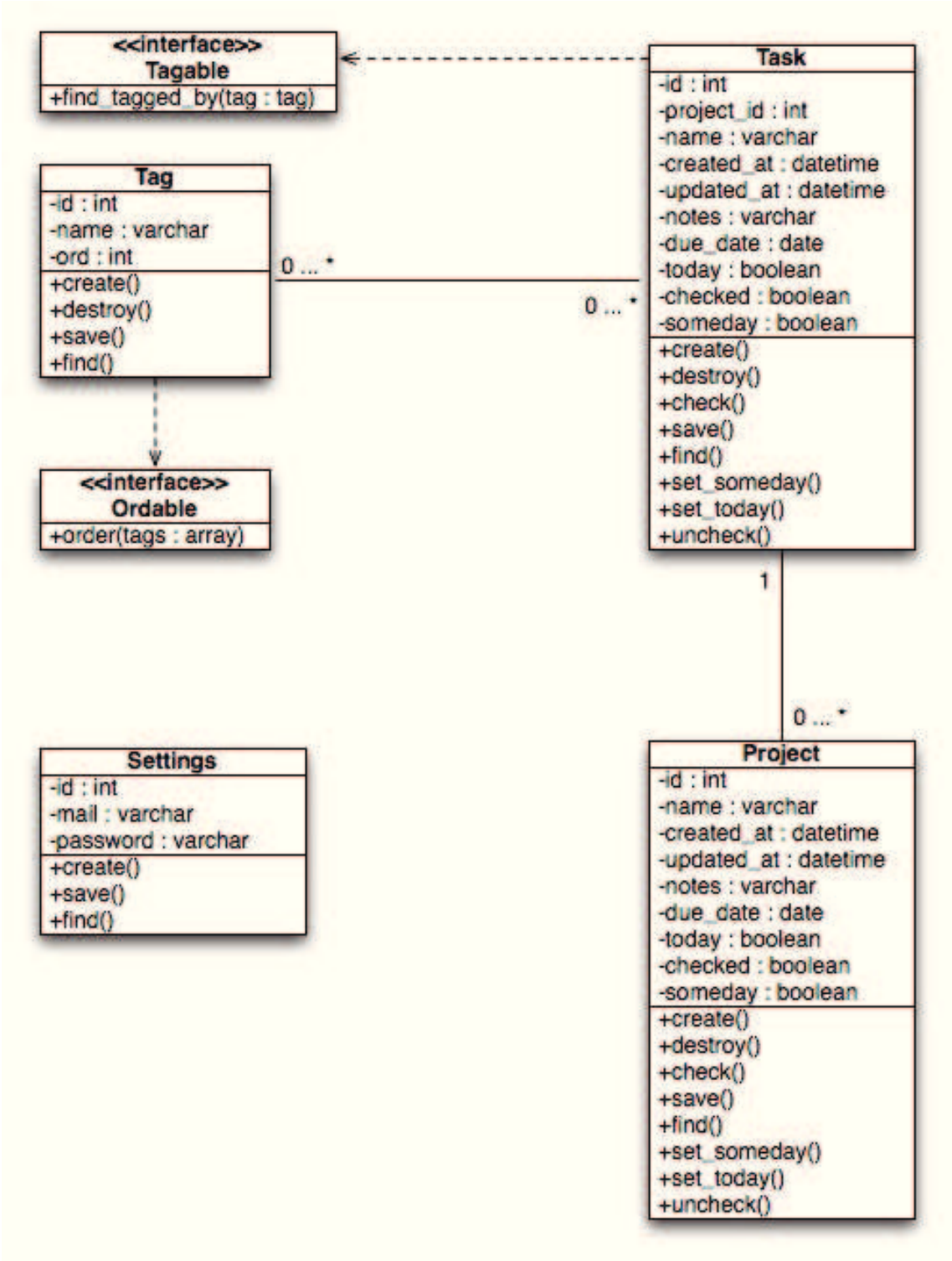
Celkovým výstupem analýzy je kompletní UML diagram viz. obrázek 8, ve kterém jsou znázorněny jednotlivé třídy, jejich vazby, metody a atributy.



Obrázek 6: Use case diagram



Obrázek 7: Konceptuální model



Obrázek 8: Výsledný konceptuální model

5 Implementace

Jak jsem již uvedl v kapitole 3, celá aplikace je implementována v jazyce Python za pomocí frameworku Django. Jelikož framework tvoří většinu kostry celé aplikace, je vhodné se s ním blíže seznámit.

5.1 Django framework

Django je ucelený framework pro efektivní vývoj webových aplikací. Instalace probíhá jednoduše stažením příslušné aktuální verze balíčku ze stránek projektu [4] . Součástí frameworku je mimo jiné rychlý development server, díky kterému vývojáři odpadají starosti s nastavováním vlastního apache serveru, tak jak tomu je při nasazování do produkčního prostředí. Další nespornou výhodu Django frameworku je rozmanitost příkazů pro generování kódu. Většina webových aplikací je z pohledu nižší vrstvy velice podobná a hierarchie aplikace jako takové se příliš nemění. Tohoto faktu samozřejmě využívají právě frameworky, které tak vývojářům dokáží nabídnout ucelenou škálu skeletonů, které lze použít okamžitě bez nutnosti téměř jakýchkoliv úprav, popřípadě minimálních. Tento proces probíhá prakticky pouze dvěma způsoby. V prvním případě, který je často k vidění především u PHP frameworků jako je např. české Nette [12] jsou skeletony dodány pouze ve formě předpřipravené struktury, do které má vývojář možnost začít doplňovat svůj kód, smazat částí, které pro něho nejsou zapotřebí apod. Druhý způsob, který je k vidění právě u Djanga a frameworku Ruby on Rails spočívá ve využití klasické unixové konzole (v případě unix based systému) nebo příkazové řádky (v případě systému Windows) pro generování kódu. Spolu s frameworkem je tak vývojáři dodána sada příkazů a nástroj, který tyto příkazy dokáže zpracovat a na jejich základě vygenerovat odpovídající kostru kódu. Ukázku tohoto způsobu demonstruje výpis 1 .

```
1 Daniels−MacBook−Pro:Projekty danieldimitrov$ django−admin.py startproject timemanagment
2
3 timememanagment/
4       init .py
5       manage.py
6       settings .py
7       urls .py
```

Výpis 1: Ukázka generování nového projektu za pomocí příkazu startproject

Pomocí příkazu startproject se generují veškeré nově vzniklé projekty v Django frameworku. Příkaz vytvoří základní strukturu, skládající se ze souborů:

- init.py - prázdný soubor sdělující Pythonu, že se jedná o složku, která budou součástí Python balíčků.
- manage.py - soubor zajišťující veškerou funkcionalitu pro zpracovávání příkazů zadaných pomocí konzole.

- settings.py - soubor pro konfiguraci projektu, obsahující základní údaje jako je nastavení časové zóny, jazyka, ale také např. přístupu k databázi a seznamu všech instalovaných aplikací, ze kterých se projekt skládá. Opravdu robusní projekty se skládají z většího množství jednotlivých aplikací, které oddělují logické části projektu. Jsou samostatné a nezávislé na ostatních aplikacích.
- urls.py - soubor obsahující seznam všech dostupných URL adres projektu, spolu s odkazem na aplikaci, controller a metodu, která zajišťuje příslušnou logiku. Množství těchto adres samozřejmě roste s velikostí projektu. V mém případě soubor nakonec obsahoval výčet cca 40ti URL adres, které byly v rámci projektu využívány. V případě, že aplikace nenalezne požadovanou URL adresu v tomto seznamu je ve vývojovém prostředí vrácena chyba ViewDoesNotExist, v případě produkčního serveru pak standardní kód HTTP 404 - File not found.

Po vytvoření této základní struktury a konfiguraci základních vlastností projektu je zapotřebí vytvořit první aplikaci pomocí příkazu startapp, tak jak demonstruje výpis 2.

```
1  Daniels-MacBook-Pro:timemanagment danieldimitrov$ python manage.py startapp projects
2
3  projects/
4      __init__.py
5      models.py
6      tests.py
7      views.py
```

Výpis 2: Ukázka generování nové aplikace projektu za pomocí příkazu startapp

Příkaz vygeneruje adresář s názvem projektu, již známý soubor init.py , soubory models.py a tests.py, které obsahují veškeré modely a testy dané aplikace a nakonec soubor views.py, který reprezentuje controller aplikace. Django nám tedy vygenerovalo dvě ze tří základních částí z MVC architektury. Pro bližší pochopení je vhodné si tyto části detailně popsat.

5.2 Modely aplikace

Soubor models.py aplikace vytvořené v předchozí kapitole 5.1, která je demonstrována výpisem 2, zapouzdřuje veškeré modely, které jsou v aplikaci použity. Django přistupuje k modelům v porovnání s ostatními frameworky z pohledu adresářové a souborové struktury vcelku netradičně. Soubor models.py standardně obsahuje všechny třídy jednotlivých modelů aplikace. Standardní architektura používána v jiných frameworcích je obvykle taková, že je pro každou třídu vytvořen zvláštní soubor, který je umístěn např. ve vytvořeném adresáři models. Vývojář má samozřejmě možnost tuto strukturu změnit, dle konvencí se to ovšem nedoporučuje. Každá tato třída je dále potomkem třídy django.db.models.Model, která zajišťuje základní funkcionalitu modelů, komunikaci s databází a ORM. Díky konvenci, kdy je jméno třídy rovno jménu entity v databázi, Django dokáže vyhodnotit k jaké entitě se daný model vztahuje.

Poznámka 5.1 Ve výpisu 3 je zobrazena ukázka třídy Projects. Na řádcích 3-10 dochází k samotné deklaraci třídy a specifikaci objektů, které odpovídají jednotlivým sloupcům entity. Na řádcích 12-15 poté dochází k deklaraci statické metody této třídy, která je dále využívána v controlleru aplikace.

```
1  from django.db import models
2
3  class Project(models.Model):
4      name = models.CharField(max_length=200)
5      created_at = models.DateTimeField()
6      updated_at = models.DateTimeField()
7      notes = models.TextField(null=True)
8      due_date = models.DateField(null=True)
9      today = models.BooleanField(default=False)
10     checked = models.BooleanField(default=False)
11
12     @staticmethod
13     def get_projects_list():
14         projects = Project.objects.filter(checked=False).order_by('-created_at')
15         return projects
16
17 class Task(models.Model):
18     project = models.ForeignKey(Project,null=True)
19     name = models.CharField(max_length=200,null=True)
20     ...
21     ...
```

Výpis 3: Ukázka modelu

5.3 Controller aplikace

Podobně jako v případě modelů je situace okolo controllerů v Djangu hodně nestandardní. Controller celé aplikace je opět součástí pouze jediného souboru a to views.py. V tomto případě se nejedná o jednotlivé třídy v rámci jednoho souboru, ale ke každému view odpovídá jedna jediná Python metoda, která daný pohled obsluhuje.

Poznámka 5.2 Ve výpisu 4 je zobrazena ukázka dvou metod obsluhující projekty v rámci aplikace, které využívají mimo jiné model Projects, který byl představen ve výpisu 3. Na řádcích 1-10 dochází k importu potřebných tříd a metod, které jsou v celém controlleru využívány. Na řádku 3 si například můžeme povšimnout importu veškerých modelů aplikace. Na řádcích 12-23 dochází k deklaraci a definování logiky metody, která zajišťuje zobrazení konkrétního projektu aplikace. Na řádcích 25-35 je dále definována metoda pro vytvoření nového projektu v aplikaci.

```
1  from django.shortcuts import render_to_response, get_object_or_404
2  from projects.models import Project, Task, Tag, TagToTask, Settings
3  from projects.google_cal import GoogleCal
4  from django.core.urlresolvers import reverse
5  from django.http import HttpResponseRedirect, HttpResponse
```

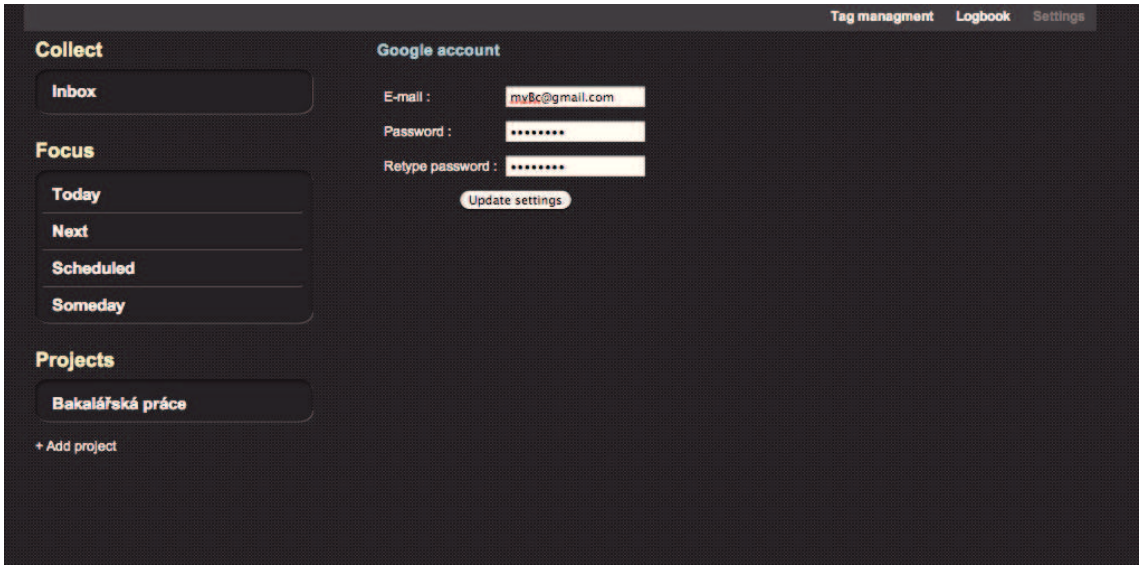
```
6  from django.template import RequestContext
7  from django.utils import simplejson
8  from inspect import stack, getmodule
9  from django.db.models import Max
10 import datetime
11
12 def show_project(request, project_id):
13     project = get_object_or_404(Project, pk=project_id)
14     tasks = project.task_set.filter(checked=False)
15     tag_to_task = TagToTask.objects.all()
16     backlink = get_current_view(request)
17     tags = Tag.objects.all().order_by('ord')
18     set_backlink(request, backlink)
19     res_tasks = TagToTask.get_tasks_by_tag( get_filter (request),tasks)
20
21     return render_to_response('show_project.html',{'project':project, 'tasks':res_tasks,
22                     'tags':tags, ' current_filter ': get_filter (request), 'task_to':tag_to_task,
23                     'projects':Project. get_projects_list ()}, context_instance=RequestContext(
                           request))
24
25 def new_project(request):
26     res = request.POST
27     if res['due_date']:
28         p = Project(name=res['name'],notes=res['notes'],due_date=res['due_date'],
29                 created_at=datetime.datetime.now(),updated_at=datetime.datetime.now(),
30                 today=res["today"])
31     else:
32         p = Project(name=res['name'],notes=res['notes'],created_at=datetime.datetime.now(),
33                     updated_at=datetime.datetime.now(),today=res["today"])
34
35     p.save()
36     return HttpResponseRedirect(reverse('projects.views.show_project', args=(p.id,)))
```

Výpis 4: Ukázka controlleru

5.4 Views aplikace

Views aplikace tvoří samostatné HTML soubory, které jsou dle konvencí zavedených v Djangu umístěny v kořenovém adresáři aplikace ve složce templates. Každé view je zodpovědné pouze za dvě věci a to vrácení HttpResponse objektu, který obsahuje obsah požadované stránky nebo zaslání příslušné výjimky. V jednotlivých views lze využívat Django template systém pro efektivní zápis Python kódu přímo do view. View standardně nevykonává žádnou logiku, ale je vhodné v něm např. pomocí template systému procházet jednotlivé prvky v poli, které bylo view předáno ze strany controlleru a tyto vypisovat jako jednotlivé řádky tabulky apod. Aby docházelo k dodržení DRY principu, je možné pro aplikaci definovat tzv. layout a v něm definovat jednotlivé bloky. Layout je ve své podstatě kostra všech view dané aplikace, které jej mohou, či nemusí využívat. Každá aplikace může mít několik takových layoutů. Při vytvoření nového view poté lze pomocí klauzule extends aplikaci oznámit jaký layout hodlá dané view využít.

Layout lze dále dělit na jednotlivé bloky. Princip je velice jednoduchý. Na jakémkoliv místě layoutu je možné definovat blok s určitým názvem. Pokud nastane situace, kdy bude zobrazováno view, které se odkazuje na takovýto layout a toto view bude obsahovat stejně definovaný blok, za kterým následuje určitý HTML kód, bude tento HTML kód vložen do příslušného layoutu a to do místa, kde je blok definován. Celkovou problematiku konkrétně popisují výpisy 5 a 6, totožné view je poté předmětem obrázku 9.



Obrázek 9: View pro nastavení přístupových údajů ke Google účtu

Poznámka 5.3 Ve výpisu 5 jsou zobrazeny dvě rozdílné části layoutu dané aplikace. V první části, je na řádcích 3-8 deklarována klasická HTML hlavička, jejíž součástí je také tag title, do kterého je vložen pomocí Django template sytému title block. Díky této deklaraci lze velice jednoduše dynamicky měnit title dané stránky, dle toho jaké view je zrovna zobrazenom tak jak demonstruje první část výpisu 6. Ve druhé části je na řádcích 13-26 zobrazeno klasická ukázka for cyklu aplikovaného pomocí Django template systému. Důležitou součástí je také block content, který dle konvencí v Djangu vyobrazuje vždy hlavní část těla daného view.

```
1   ...
2   ...
3   <meta http-equiv="Expires" content="never" />
4   <meta name="revisit-after" content="7 days" />
5   <meta name="distribution" content="Global" />
6   <title>{% block title %}{% endblock %}</title>
7   <link type="text/css" href="/static/css/reset.css" rel="Stylesheet" />
8   <link type="text/css" href="/static/css/style.css" rel="Stylesheet" />
9   ...
10  ...
11                      ...
```

```
12                  ...
13                  {% for project in projects %}
14                      {% if forloop.first %}
15                          <li class="droppable" name="project" value="{{project.id}}">
16                              <a href="/projects/{{project.id}}/">{{project.name}}</a>
17                          </li>
18                      {% else %}
19                          <li class="sep">
20                              <img src="/static/css/images/sep.png" width="234px" height="3px">
21                          </li>
22                          <li class="droppable" name="project" value="{{project.id}}">
23                              <a href="/projects/{{project.id}}/">{{project.name}}</a>
24                          </li>
25                      {% endif %}
26                  {% endfor %}
27              </div>
28              <div id="cat_bottom"></div>
29          </div>
30          <br>
31          <a href="/projects/add_project/">+ Add project</a>
32      </div>
33      {% block content %}{% endblock %}
34  </div>
35  ...
36  ...
```

Výpis 5: Ukázka layoutu aplikace

Poznámka 5.4 Ve výpisu 6 je zobrazeno view umožňující uživateli nastavit údaje ke svému Google účtu pro potřeby kalendáře v aplikaci. Na řádku 1 je použita klauzule extends, pro definici layoutu, které toto view využije. Na řádku 2 je poté pomocí blocku title nastaven titulek stránky. Řádky 4-30 tvoří pak formulář, který představuje hlavní část celého view, která je logicky umístěna do blocku content.

```
1  {% extends "layout.html" %}
2  {% block title %}Settings{% endblock %}
3
4  {% block content %}
5      <span class="project_name">Google account</span>
6      <br><br>
7      <form action="/projects/update_settings/" method="post">
8          {% csrf_token %}
9          <table>
10             <tr>
11                 <td>E-mail :</td>
12                 <td><input type="text" name="mail" id="mail" value="{%␣for␣set␣in␣settings␣%}{{set.
                       mail}}{%␣endfor␣%}"></td>
13             </tr>
14             <tr>
15                 <td>Password :</td>
16                 <td><input type="password" name="password" id="password"></td>
17             </tr>
```

```
18              <tr>
19                  <td>Retype password :</td>
20                  <td><input type="password" name="password2" id="password2"></td>
21              </tr>
22              <tr>
23                  <td colspan="2" style="text-align:center">
24                      <input type="submit" name="add" value="Update settings">
25                  </td>
26              </tr>
27          </table>
28          <input type="hidden" name="id" id="id" value="{% for set in settings %}{{set.id}}{% endfor
                %}">
29      </form>
30  {% endblock %}
```

Výpis 6: Ukázka view pro settings

5.5 Google Calendar

Jednou ze stěžejních částí celé aplikace byla implementace kalendáře. Jednou z nejrozšířenějších služeb v této kategorii je služba Google Calendar [13], která je zdarma dostupná k jakémukoliv Google účtu. Díky rozšířenosti a vyhovujícím parametrům služby, mezi které patří také veřejné API, jsem se na základě domluvy s garantem mé bakalářské práce Ing. Janem Gaurou rozhodl tuto službu do aplikace částečně integrovat. Cílem implementace bylo získání dat z kalendáře, interpretace těchto dat v systému a setřídění dle časové posloupnosti, spolu s ostatními záznamy v systému na příslušných listech. Pro implementaci této funkcionality bylo zapotřebí prostudovat a implementovat komunikaci prostřednictvím Google Calendar API [14].

5.5.1 Google Calendar API

Kompletní dokumentace ke Google Calendar API je standardně veřejně dostupná pro jakékoliv vývojáře na stránkách projektu [14]. Prvním krokem, který je nutné splnit pro implementaci komunikace s tímto API je instalace příslušných Python balíčků a jejich integrace do systému.

5.5.1.1 Instalace API Instalace Google Calendar API probíhá ve dvou základních krocích:

- Stažení a instalace potřebných součástí - Pro práci s Google Data services je zapotřebí do aplikace integrovat balíček ElementTree [15] pro efektivní práci s XML soubory a knihovnu Google Data Library [16], která zapouzdřuje základní logiku potřebnou pro komunikaci s Google Data services. ElementTree je nutno instalovat v případě Pythonu verze 2.5. a nižší. Od vyšších verzí je balíček standardně součástí Python distribuce, což byl také případ této bakalářské práce, kdy byl v průběhu instalace Django frameworku nainstalován Python verze 2.6.1. Instalace

Google Data Library probíhá jednoduše stažením příslušného archivu, jeho rozbalením a zkopírováním adresářů gdata a atom do kořenového adresáře projektu.

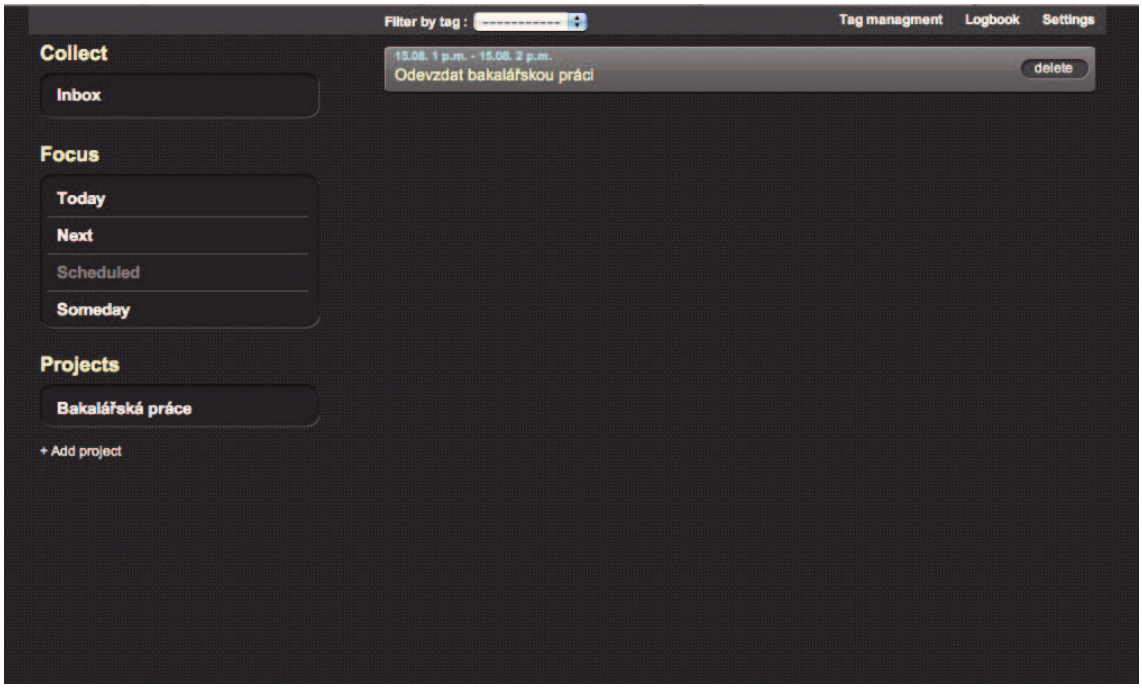
- Integrace nových částí do projektu - Po stažení a instalaci nových součástí, je potřeba tyto části integrovat přímo do aplikace. Tento krok probíhá importem standardních Python balíčků a je demonstrován výpisem kódu 7, který je umístěn v modelu zajišťujícím komunikaci s Google Data services.

```
1 from xml.etree import ElementTree
2 import gdata.calendar.service
3 import gdata.service
4 import atom.service
5 import gdata.calendar
```

Výpis 7: Ukázka importu balíčků potřebných pro komunikaci s Google Data services

5.5.1.2 Implementace API v systému

Pro účely komunikace se službou Google Calendar byl v systému vytvořen samostatný model, který komunikaci kompletně zapouzdřuje. Model implementuje dvě základní metody pro získání samotného objektu kalendáře a získání jednotlivých událostí kalendáře, dále pomocnou metodu pro setřídění těchto událostí dle časové posloupnosti. Jednotlivé metody demonstrují výpisy kódu 8, 9 a 10. Obrázek 10 poté demonstruje view pro kalendář v aplikaci, které dané metody využívá.



Obrázek 10: View znázorňující jednotlivé události importované z Google kalendáře

Poznámka 5.5 Výpis 8 popisuje metodu pro získání kalendáře, náležícího konkrétnímu Google účtu.

```
1  def get_calendar_service(mail, password):
2      calendar_service = gdata.calendar.service.CalendarService()
3      calendar_service.email = mail
4      calendar_service.password = password
5      calendar_service.source = 'Google-Calendar_Python_Sample-1.0'
6      calendar_service.ProgrammaticLogin()
7
8      return calendar_service
```

Výpis 8: Ukázka metody pro získání objektu kalendáře

Poznámka 5.6 Výpis 9 popisuje metodu pro získání jednotlivých událostí konkrétního kalendáře. Na řádcích 2-10 je implementováno úvodní nastavení v závislosti, pro který z listů v systému mají být data získána. Řádky 12-47 obstarávají základní logiku celé metody. V první části jsou získány veškeré události z kalendáře za konkrétní období, tyto informace jsou dále zpracovávány do vyhovující podoby a nakonec vráceny v podobě pole.

```
1  def get_calendar_events(calendar_service, start):
2      if start == 6:
3          start_date = datetime.date.today()
4          end_date = str(start_date + datetime.timedelta(180))
5          start_date = str(start_date)
6
7      if start == 1:
8          start_date = datetime.date.today()
9          end_date = str(start_date + datetime.timedelta(1))
10         start_date = str(start_date)
11
12     query = gdata.calendar.service.CalendarEventQuery('default', 'private', 'full')
13     query.start_min = start_date
14     query.start_max = end_date
15     feed = calendar_service.CalendarQuery(query)
16     events = []
17     for i, an_event in enumerate(feed.entry):
18         f_link = an_event.GetEditLink().href
19         s_link = string.split(f_link,"/")
20         link = s_link[len(s_link)-2]+"_"+s_link[len(s_link)-1]
21         title = an_event.title.text
22         for a_when in an_event.when:
23             start_time = a_when.start_time
24             end_time = a_when.end_time
25             if string.find(a_when.start_time,"T") > -1:
26                 print(a_when.start_time)
27                 st_date = string.split(start_time,"T")
28                 st_time = string.split(st_date[1],".")
29
30                 start_date = datetime.date(int(string.split(st_date[0],"-")[0]),
```

```
31                                  int(string.split(st_date[0],"-")[1]),
32                                  int(string.split(st_date[0],"-")[2]))
33
34              start_time = datetime.time(int(string.split(st_time[0],":")[0]),
35                                  int(string.split(st_time[0],":")[1]),0,0)
36
37              en_date = string.split(end_time,"T")
38              en_time = string.split(en_date[1],".")
39
40              end_date = datetime.date(int(string.split(en_date[0],"-")[0]),
41                                  int(string.split(en_date[0],"-")[1]),
42                                  int(string.split(en_date[0],"-")[2]))
43
44              end_time = datetime.time(int(string.split(en_time[0],":")[0]),
45                                  int(string.split(en_time[0],":")[1]),0,0)
46
47              events.append([title,start_date,start_time,end_date,end_time,link])
48          else:
49              st_date = string.split(start_time,"-")
50              start_date = datetime.date(int(st_date[0]),int(st_date[1]),int(st_date[2]))
51              en_date = string.split(end_time,"-")
52              end_date = datetime.date(int(en_date[0]),int(en_date[1]),int(en_date[2]))
53              events.append([title,start_date,'-',end_date,'-',link])
54
55      res = GoogleCal.sort_events_by_date(events)
56      return res
```

Výpis 9: Ukázka metody pro získání jednotlivých událostí kalendáře

Poznámka 5.7 Výpis 10 popisuje metodu pro setřídění událostí dle časové posloupnosti.

```
1   def sort_events_by_date(events):
2       max = len(events)
3       for i in range(0,max):
4           for j in range(1,max-i):
5               if events[j-1][1] == events[j][1]:
6                   if not events[j][2] == '-':
7                       if events[j-1][2] > events[j][2]:
8                           temp = events[j-1]
9                           events[j-1] = events[j]
10                          events[j] = temp
11              else:
12                  if events[j-1][1] > events[j][1]:
13                      temp = events[j-1]
14                      events[j-1] = events[j]
15                      events[j] = temp
16
17      return events
```

Výpis 10: Ukázka metody pro setřídění událostí kalendáře

6 Další možná rozšíření

Aplikaci byla naprogramována tak, aby byla v budoucnu co nejjednodušeji rozšiřitelná o další části. V následujících podkapitolách si proto dovolím navrhnout několik základních rozšíření.

6.1 Větší počet uživatelů

Aplikace momentálně slouží jako regulérní time managmentový systém pro jednoho uživatele. V budoucnu by bylo zřejmě vhodné aplikaci rozšířit o takové funkce, aby mohla být využívána větším počtem uživatelů a umístit ji na dostupný veřejný, či intranetový server. Pro toto rozšíření bude zapotřebí implementovat následující úpravy:

- Připravit view pro přihlášení současného a registraci nového uživatele.
- Připravit metody v controlleru aplikace, které budou dané view obsluhovat. Dále přidat metodu pro kontrolu identity uživatele, která by měla být uložena po přihlášení v session.
- Datovou strukturu bude nutné obohatit o tabulku User, skládající se z id, uživatelského jména a hesla uživatele. Dále bude nutné do entit Tag, Task, Project a Settings přidat atribut userId tak aby bylo zřejmé, kterému uživateli daný záznam patří. Tento atribut bude vhodné indexovat.
- Současné metody obsluhující jednotlivá view v controlleru respektive modelu aplikace bude nutné upravit tak, aby přistupovaly a modifikovaly data pouze přihlášeného uživatele.

6.2 Rozšíření o další formáty kalendáře

V současné chvíli dokáže aplikace přistupovat pouze ke kalendáři prostřednictvím Google účtu. Jednotlivých služeb a formátů, které zajišťují podobné služby, je samozřejmě mnoho. V budoucnu, by proto bylo zřejmě vhodném rozšířit kalendář o možnost importu kalendářů dalších služeb třetích stran, jako je například iCal, či kalendáře dostupné na Microsoft Exchange Serverech apod. Pro tyto rozšíření bude zapotřebí implementovat přibližně následující kroky:

- Získat API specifikaci k dané službě, popřípadě konkrétní popis souboru pro import.
- View settings rozšířit o možnost nastavení formátu, popřípadě služby, ze které má být kalendář importován do systému. V případě formátu, pak přidat možnost výběru souboru pro import.
- Rozšířit modely aplikace o metody zajišťující přístup k API těchto služeb, popřípadě rozšířit controller aplikace o metody zpracovávající importy v daných formátech.

- Modifikovat současné metody obsluhující view settings, tak aby zajišťovaly vše potřebné.
- Přidat do entity settings atributy services, popřípadě formats typu enum a specifikovat jednotlivé služby a formáty, které budou uživateli k dispozici.

6.3 Pokročilejší práce v týmu

Týmová spolupráce v rámci GTD je ožehavé téma. Celý problém spočívá v tom, že GTD je systémem pro osobní time managment. V případě menších, či větších týmů by bylo samozřejmě žádoucí GTD posunout trošku dále a dokázat tuto metodiku aplikovat např. částečně v projektovém řízení. Nicméně žádný systém tohoto druhu zatím neexistuje a každý pokus o takovou implementaci skončil fiaskem. V současné verzi systému je práce v týmu a delegování řešeno pomocí kontextů a listu čekám na, kde má uživatel možnost zaznamenat úkoly, jejichž zpracováním pověřil jiné uživatele a u těchto úkolu nastavit např. datum splnění, při kterém má být upozorněn na to, že by měl danou věc urgovat. Bohužel se jedná o pracnou záležitost. Uživatel musí daný závazek zaznamenat a následně jej ještě předat konkrétní osobě. Jedním ze špatných řešení by bylo vytvořit týmovou architekturu, kde by nadřízení pracovníci měli možnost vkládat závazky podřízeným pracovníkům do vstupní schránky. Tento model se na základě mnoha pokusů osvědčil jako nevhodný. V systému při větším množství těchto požadavků začne vznikat zmatek a stává se nepřehledným. Daleko rozumnějším řešením je implementovat do systému možnost zasílání těchto požadavků do oddělené schránky a implementovat notifikace, které na novou zprávu uživatele upozorní. V praxi by systém poté mohl fungovat tak, že by nebylo nutné nastavovat hierarchii v rámci týmu, ale bylo by nutné pouze specifikovat jednotlivé týmy, respektive okruhy lidí, kteří by měli možnost si různé požadavky navzájem zasílat. Při zadávání konkrétního požadavku na list čekám na by systém nabídl zadavateli možnost specifikovat uživatele, kterým má být požadavek zaslán do příslušné schránky. Takovýmto uživatelům, by byla poté v systému zobrazena notifikace, upozorňující na nový požadavek konkrétní osoby a měly by možnost si jej podobně jako požadavek e-mailem zpracovat do své vstupní schránky a zaznamenat si k němu vlastní formulace a poznámky.

7 Závěr

Úkolem mé bakalářské práce byla implementace systému osobního time managmentu a produktivity práce dle metodiky Getting Things Done, jejímž autorem je americký kouč a expert v této oblasti David Allen.

V úvodu bylo nutné detailně prostudovat celou metodiku a určit části celé koncepce, které bylo možné systematizovat. Dále bylo nutné detailně prostudovat základní části metodiky - zaznamenávání, objasňování, organizování, reflexi, akci a z důvodu pochopení všech detailů se je pokusit osvojit.

V průběhu práce bylo nutno se seznámit s technologiemi a metodami, které využívá Django framework, osvojit si a detailně pochopit návrhový vzor MVC.

Na základě těchto vědomostí bylo zapotřebí analyzovat celkovou náročnost projektu, veškeré vstupní požadavky a navrhnout koncepci systému, který všem těmto kritériím vyhovuje. Součástí této analýzy byla analýza funkčních a nefunkčních požadavků na systém, analýza logistiky celého systému, datová analýza a jejím výstupem byl konceptuální model znázorňující konečnou strukturu.

Poslední částí byla implementace, jejíž stěžejní části byla integrace služby Gooogle Calendar. Pro tyto účely bylo nutné prostudovat Google Calendar API a za pomocí knihovny Google Data Library zajistit komunikaci s Google Data services.

Závěrem byly poté navrhnuty další možnosti rozšíření systému a koncepty, které by bylo vhodné do systému do budoucna implementovat.

Výsledný systém, dle mého názoru splňuje veškeré požadavky, které na něj byly kladeny, a je vyhovujícím řešením pro správu osobního time managmentu dle metodiky Getting Things Done.

Daniel Dimitrov

8 Reference

[1] Covey, S. R.: *7 návyků skutečně efektivních lidí,* Management Press, 2007, ISBN 80-7261-156-9

[2] Allen, D.: *Mít vše Hotovo, Jak zvládnout práci i život a cítit se přitom dobře,* Jan Melvil Publishing, 2008, ISBN 978-80-903912-8-4

[3] Allen, D.: *Aby vše klapalo, Jak hravě zvládnout pracovní i životní výzvy,* Jan Melvil Publishing, 2009, ISBN 978-80-87270-00-4

[4] Django, [online], [cit. 2011-08-12]. URL *https://www.djangoproject.com/*

[5] Bernard, B.: Úvod do architektury MVC, [online], 2009-05-07 [cit. 2011-08-07]. URL *http://zdrojak.root.cz/clanky/uvod-do-architektury-mvc/*

[6] Bernard, B.: Prezentační vzory z rodiny MVC, [online], 2009-05-11 [cit. 2011-08-07]. URL *http://zdrojak.root.cz/clanky/prezentacni-vzory-zrodiny-mvc/*

[7] GMail, [online], [cit. 2011-08-13]. URL *http://mail.google.com*

[8] Ruby on Rails, [online], [cit. 2011-08-13]. URL *http://rubyonrails.org/*

[9] Cocoa, [online], [cit. 2011-08-13]. URL *http://developer.apple.com/technologies/mac/cocoa.html*

[10] Hegarty, P.: Developing Apps for iOS (HD), [iTunes], 2010 [cit.2011-07-18]. URL *http://itunes.apple.com/us/itunes-u/developing-apps-for-ios-hd/id395605774*

[11] SQLite, [online], [cit. 2011-08-13]. URL *http://www.sqlite.org/*

[12] Nette Framework, [online], [cit. 2011-08-13]. URL *http://nette.org/*

[13] Google Calendar, [online], [cit. 2011-08-13]. URL *http://www.google.com/calendar*

[14] Google Calendar API, [online], [cit. 2011-08-14]. URL *http://code.google.com/intl/cs/apis/calendar/*

[15] ElementTree, [online], [cit. 2011-08-14]. URL *http://effbot.org/zone/element-index.htm*

[16] Google Data Library, [online], [cit. 2011-08-14]. URL *http://code.google.com/p/gdata-python-client/downloads/list*

A Uživatelská dokumentace

Uživatelská dokumentace popisující uživatelské možnosti celé aplikace je součástí přiloženého kompaktního disku ve formátu PDF a nachází se v souboru dokumentace.pdf .